JS interview questions

1.Why let and const were introduced when var already existed?

var had **problems**:

- Function-scoped (not block-scoped)
- Allows redeclaration
- Causes bugs due to hoisting confusion
-

let and const fixed this:

- **Block-scoped** ({})
- No redeclaration in same scope
- const prevents reassignment

---

2. How does JavaScript decide the data type of a variable at runtime?

JavaScript is **dynamically typed**:

- Type is decided **when value is assigned**, not when variable is declared

```
let x = 10;      // number
x = "hello";    // string
```

---

3. Difference between == and ===, and why companies prefer ===?

Operator	**Comparison**
==	Value only (type conversion)
===	Value + Type (strict)

```
5 == "5"   // true
5 === "5"  // false
```

- Avoids unexpected bugs
- No implicit type conversion
- More predictable behavior

---

4. How do logical operators (&&, ||) help write cleaner code?

They reduce **if-else clutter**.

```
isLoggedIn && showDashboard();
username || "Guest";
```

- && → execute if true
- || → fallback value

---

5. When to choose switch over if-else?

Use switch when:

- One variable
- Many fixed values

```
switch (day) {
  case "Mon": break;
  case "Tue": break;
}
```

6. Why does do-while execute at least once?

Because:

- Condition is checked **after** execution

```
do {
  console.log("Runs once");
} while (false);
```

---

7. What happens internally when a for loop runs?

Steps:

1. Initialization (let i = 0)
2. Condition check (i < n)
3. Execute body
4. Increment (i++)
5. Repeat

---

8. How does array indexing work? What if index doesn't exist?

Arrays use **0-based indexing**.

```
let arr = [10, 20];
arr[5]; // undefined
```

- No error
- Returns undefined

---

9. Difference between map() and forEach()?

Feature	**map()**	**forEach()**
Returns new array	yes	no
Used for	Transformation	Side effects

```
arr.map(x => x * 2);
arr.forEach(x => console.log(x));
```

---

10. Why are higher-order array methods preferred?

Because they:

- Improve readability
- Reduce bugs
- Follow functional programming
- Avoid manual index handling

Examples:

- map
- filter
- reduce

---

11. Object vs Primitive memory storage

Primitive	Object
Stored by value	Stored by reference
Immutable	Mutable

```
let a = 10;
let b = a;   // copy

let obj1 = {};
let obj2 = obj1; // same reference
```

---

12. for...in vs Object.keys()

Feature	for...in	Object.keys()
Iterates	Keys incl. inherited	Own keys only
Control	Less	More predictable

---

13. Parameters vs Arguments & Callbacks

- **Parameters** → variables in function definition
- **Arguments** → actual values passed

```
function greet(name) { } // parameter
greet("Surya");        // argument
```

Callback: function passed as argument

```
setTimeout(() => {}, 1000);
```

---

14. Why async JS doesn't block the main thread?

JavaScript uses:

- Event Loop
- Call Stack
- Web APIs

Async tasks run in background → callback runs later.

15. Role of the V8 engine in JS & React apps

V8 JavaScript Engine:

- Converts JS to machine code
- Executes JS fast
- Handles memory & garbage collection

React apps rely on V8 (in Chrome & Node.js) for:

- Fast rendering
- Efficient execution